Microsoft®
Windows™95

CURSO IBM DE PROGRAMACION

16

EN Windows 95

IBM

PLANETA DeAGOSTINI

Microsoft®
Windows™ 95

CURSO IBM DE PROGRAMACION EN Windows 95

VOLUMEN 2 - UNIDAD 16

Director Gerente: José Mª Puig de la Bellacasa
Director Técnico: Francisco Valer Bañales
Director de Marketing: Javier Graupera Gassol
Director Editorial: Artur Klein Moreno
Directora de Software: Beatriz Cerrato Vidal

Dirección y coordinación: José Luis Becerril Gasco. De la Corporación IBM España, Licenciado en Ciencias Físicas, Director del Instituto de Inteligencia Artificial de IBM España.

Redactores y colaboradores: *José Ramón Dorronsoro Ibero.* Doctor en Ciencias Matemáticas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Vicente López Martínez.* Doctor en Ciencias Químicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Manuel Alfonseca Moreno.* Doctor Ingeniero de Telecomunicación y Licenciado en Informática, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Garrido Salas.* Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Javier Martínez Rodríguez.* Doctor en Ciencias Físicas, Profesor Titular del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Santiago de Aréchaga Tarruell.* Ingeniero Superior de Informática, Investigador del Instituto de Ingeniería del Conocimiento (UAM). *Pedro Pascual Broncano.* Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid. *Carlos Aguirre Maeso.* Licenciado en Ciencias Matemáticas, Profesor Asociado del Departamento de Ingeniería Informática de la Universidad Autónoma de Madrid, Investigador del Instituto de Ingeniería del Conocimiento (UAM).

Realización editorial: LH servicios informáticos, S.L.

Diseño gráfico: Esteoeste

© Multimedia Ediciones, S.A., Barcelona

ISBN Obra Completa: 84-88720-79-3
ISBN Fascículos: 84-88720-80-7
ISBN Volumen 2: 84-88720-82-3
Depósito Legal: B-33.798-97

Redacción y administración:
Aribau 198, 5ª planta - 08036 - Barcelona
Tel. (93) 306.78.90 - Fax (93) 306.78.97

Servicio de atención al cliente: Teléfono 902.101.324

Petición de números atrasados:
Teléfono 902.288.288
Apartado de correos nº 2.867, -08080- Barcelona

Servicio de consulta:
Teléfono 902.244.444 de lunes a viernes, de 10.00 a 14.30 y de 15.30 a 19.00 horas.
Fax (91) 661.67.08 durante las 24 horas del día.

Distribuye para España: MIDESA. Marco Ibérica de Ediciones, S.A., Carretera de Irún, km. 13,350, variante de Fuencarral - 28034 - Madrid

Venta en quioscos y librerías: Pida a su proveedor que le reserve sus ejemplares del curso. Adquiriéndolos todas las semanas en el mismo quiosco o librería facilitará la distribución y obtendrá un mejor servicio. El editor se reserva el derecho de modificar el precio de venta de los componentes de la colección en el transcurso de la misma, si las circunstancias del mercado así lo exigieran.

Composición del Curso: El Curso IBM de Programación se publica en forma de 52 unidades de aparición semanal y 52 CD-ROM. Las 52 unidades se encuadernan en 5 volúmenes. Con la unidad que completa cada volumen se ponen a la venta las tapas para su encuadernación.

Multimedia Ediciones garantiza la publicación de todos los elementos que componen esta obra.

© Editorial Planeta Argentina S.A.I.C.
Independencia, 1668 - Capital Federal
Teléfono atención al cliente: 482-0655 y 482-0657
Distribuye en capital: Huesca-Sanabria. En interior: D.G.P.
© Editorial Planeta Mexicana S.A. de C.V.
Avda. Insurgentes # 1162 - México D.F.
Teléfono de atención al cliente: 575-1348
© Editorial Planeta Colombiana S.A.
Carrera 68A, nº 25-55 - Santafé de Bogotá
Teléfono de atención al cliente: 9800-15315

Imprime: Cayfosa. Carretera de Caldes, km. 3, Santa Perpetua de Mogoda (Barcelona).

INSTALACION/DESINSTALACION DEL SOFTWARE: Para la correcta instalación y desinstalación del CD-ROM que se entrega en esta unidad, véase el libreto que acompaña a cada CD-ROM.

Marcas comerciales registradas: En esta publicación son marcas registradas de sus respectivas compañías: Delphi Desktop, dBase, Turbo Pascal, Turbo C++, Turbo Assembler, Turbo Debugger, Borland Intrabuilder, Paradox, Visual dBase y BRIEF (Borland International, Inc.). Lotus, Lotus 123 y SmartIcons (Lotus Development Corporation, filial de IBM). Microsoft DOS, Microsoft Windows, Microsoft Windows 95, Excel, Visual Basic, Access y FoxPro (Microsoft Corporation). Netscape Navigator Gold (Netscape Communications Corporation). Java (Sun Microsystems, Inc.). IBM (International Business Machines).

CURSO DE PROGRAMACIÓN VISUAL: TURBO PASCAL (3)

CONTROL DEL FLUJO DE INSTRUCCIONES

Si los programas de ordenador tuvieran que ser una secuencia de instrucciones que hubiera que recorrer siempre una detrás de otra, desde el principio hasta el final, los ordenadores carecerían de la utilidad que han demostrado tener.

Desde los primeros lenguajes de programación de alto nivel, el programador tiene a su disposición en los lenguajes formas de redireccionar el flujo de instrucciones que se han de ejecutar, ya sea para repetir algunas de ellas o para realizar unas u otras dependiendo de que se cumplan determinadas condiciones.

Ésta es la razón por la que se pueden escribir programas muy cortos que harán que el ordenador realice tareas muy complejas y diferentes en cada una de las situaciones que se consideren al diseñar el programa.

INSTALACIÓN

El proceso de instalación de esta unidad sigue los mismos pasos de entregas anteriores. Insertar el CD con el programa en el lector del equipo, abrir el menú Inicio y activar la opción Ejecutar. Después, teclear D:\INSTALAR, o bien dejar que se inicie el programa Autorun automáticamente.

ESTRUCTURAS DE SELECCIÓN

En los programas que se han analizado hasta ahora, las instrucciones se realizaban una detrás de otra. Todas las órdenes provocaban algún efecto.

En realidad, ésta es una situación poco común. Es mucho más frecuente que tengan que ser distintas las tareas que ha de realizar la máquina para los diferentes casos que se planteen. El primer elemento del lenguaje necesario para realizar una bifurcación del flujo de instrucciones es la estructura de selección en la que se consigue elegir un bloque de instrucciones u otro dependiendo del valor de una variable o del resultado de operar sobre un grupo de ellas.

En el caso más sencillo se trata de decidir entre dos posibilidades. Para ello, el lenguaje Pascal ofrece la estructura **If** básica que se ha de utilizar siguiendo la siguiente sintaxis:

```
If ValorBoolean
Then accion
```

donde "ValorBoolean" es un dato del tipo Boolean o una expresión que proporciona como resultado ese tipo de dato, y "accion" es una instrucción sencilla o compuesta.
Sólo en el caso de que "ValorBoolean" tenga el valor True se ejecutará la instrucción "accion".

Una segunda opción puede especificarse con la estructura If completa:

```
If ValorBoolean
Then una
Else otra
```

donde "una" y "otra" son instrucciones que se realizarán dependiendo de cual sea el valor que tome "ValorBoolean". Si éste es True se ejecuta "una" , y si es False se ejecuta "otra". El gráfico sintáctico de la estructura If puede verse en la figura 1.

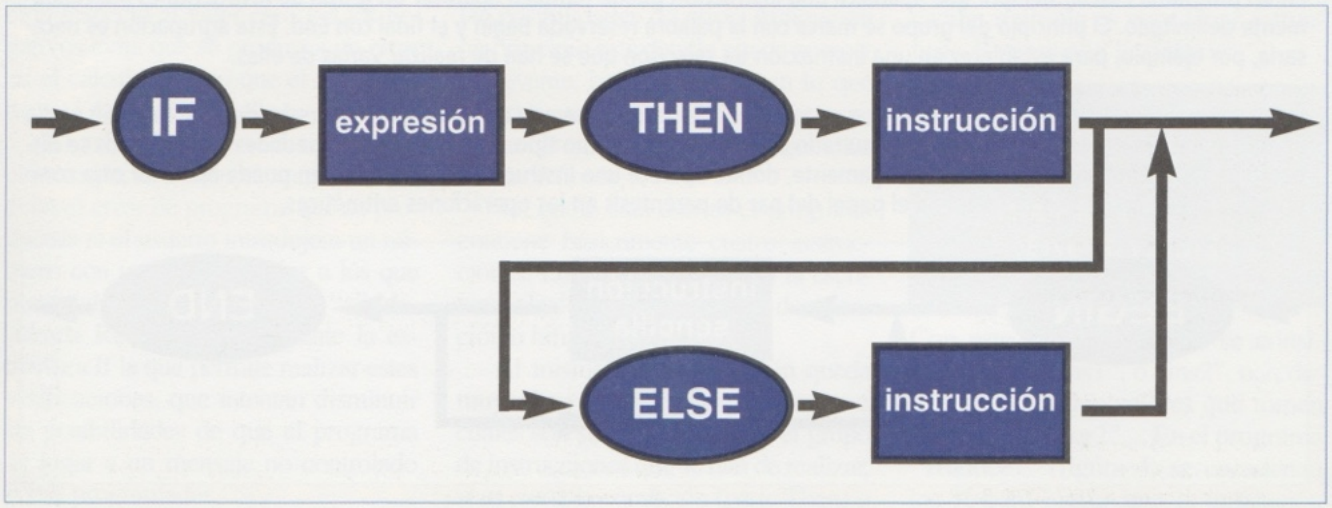


Fig. 1. Gráfico sintáctico de la instrucción If.

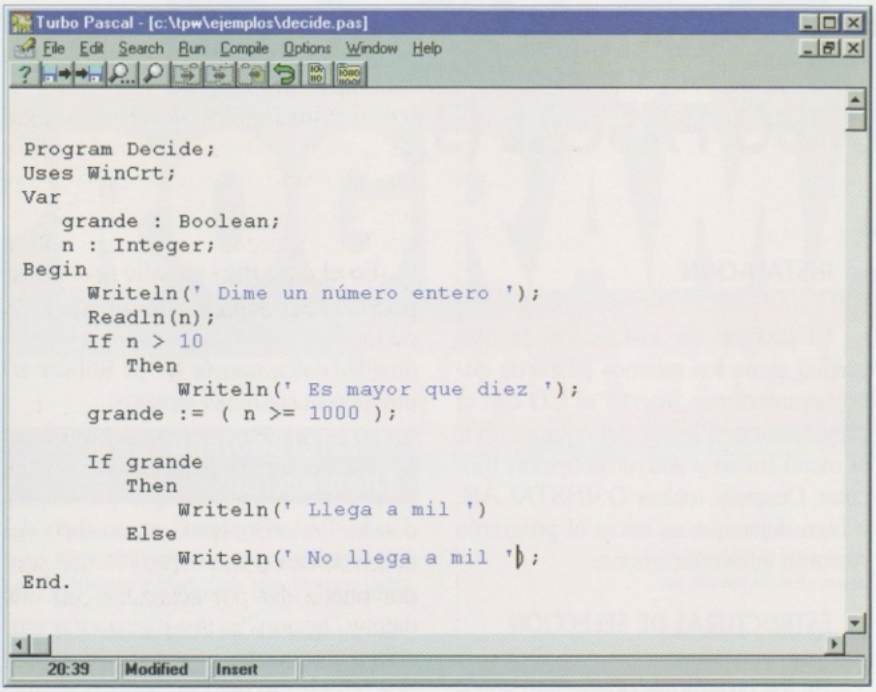


Fig. 2. El programa utiliza la estructura If para analizar el tamaño de un número.

En ambos casos, después de la estructura If se da paso al resto del programa y se ejecuta la instrucción siguiente. Si se trata de la construcción que lleva Else, se ejecutará la instrucción "una" u "otra" y luego la instrucción que venga a continuación.

La traducción al castellano de "If A Then B, Else C" podría ser "si A entonces B, de lo contrario C".

El programa "Decide" (figura 2) muestra un ejemplo sencillo de la utilización de estas estructuras If.

En la primera de ellas se utiliza para obtener un valor lógico de una condición (una elección entre dos posibilidades), mientras que en la segunda se utiliza un valor lógico (almacenado en una variable) para determinar la acción a realizar. Merece la pena detenerse en el uso de la puntuación ";" en las estructuras If. Se trata de algo sencillo, pero que tiende a olvidarse y da lugar a errores que el compilador detecta. Además, se olvida de tal modo que, cuando el compilador avisa del error, siempre se tiende a pensar que se trata de algo más sofisticado.

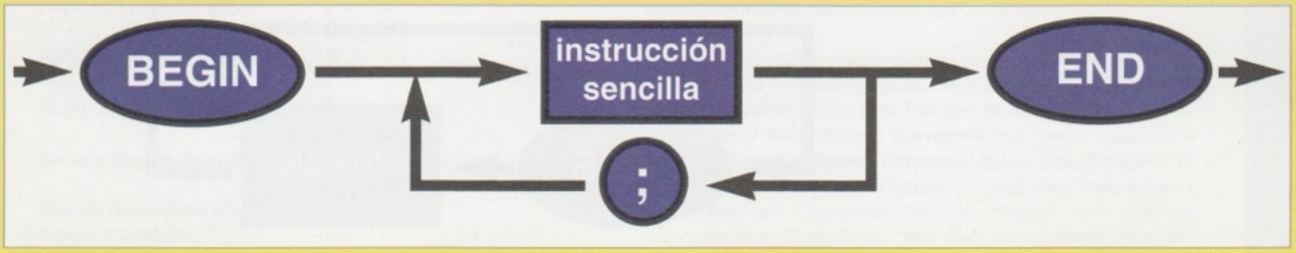
La puntuación ";" se utiliza en Pascal para indicar que una instrucción termina y, cuando está presente la parte Else en la instrucción If, ésta no acaba hasta que se ha especificado la instrucción que ha de realizarse cuando la condición no sea cierta. Por tanto, la puntuación ";" sólo debe ir después de esta instrucción. Por ejemplo, en el programa "Decide", la primera estructura If no tiene segunda parte Else, por tanto el ";" va después de la llamada a la función WriteLn. Por el contrario, no es necesario ningún ";" en la segunda estructura If, justo antes del Else. Es la propia palabra reservada Else la que indica que ya se ha terminado de especificar la instrucción que se ha de realizar cuando la condición sea cierta.

Como ejercicio práctico se recomienda escribir el programa "Decide" en Turbo Pascal, así como jugar con las puntuaciones ";" para experimentar los avisos que el compilador proporciona cuando se sitúan en el lugar equivocado. También es conveniente escribir, partiendo de cero, otros programas en los que se utilice la estructura If de modo que pueda asimilarse bien el uso de esta estructura.

INSTRUCCIONES COMPUESTAS

En un programa Pascal, siempre que aparezca una instrucción puede también aparecer un grupo de instrucciones adecuadamente delimitado. El principio del grupo se marca con la palabra reservada Begin y el final con End. Esta agrupación es necesaria, por ejemplo, para establecer en una instrucción de selección que se han de realizar varias de ellas.

En las expresiones aritméticas se estudió que en todo lugar en el que aparecía un dato de un determinado tipo, también podía aparecer una operación que diera como resultado un dato de ese mismo tipo. Para evitar ambigüedades en la sintaxis se disponía de los paréntesis. Pues bien, análogamente, donde aparece una instrucción simple también puede aparecer otra compuesta y el par Begin-End desempeña el papel del par de paréntesis en las operaciones aritméticas.



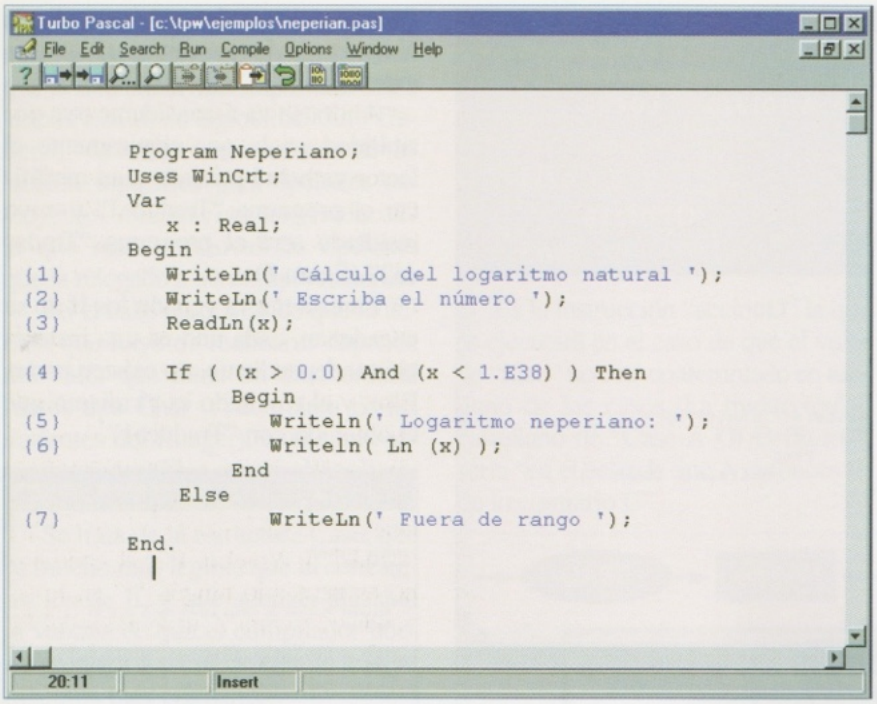


Fig. 3. Uso de instrucciones compuestas en una verificación con la estructura If.

Las acciones que se ejecutan en la estructura If pueden ser sencillas o compuestas.

El programa "Neperiano" (figura 3) es un ejemplo del uso de la estructura If con una instrucción compuesta.

Se trata de un programa que sencillamente calcula el logaritmo natural del número que teclea el usuario. Pero en este programa hay una diferencia fundamental con los que se podían escribir hasta ahora para el cálculo de funciones matemáticas. "Neperiano" intenta cumplir los requisitos de los programas profesionales. La verificación respecto al valor cero y a los negativos evita que se produzca un error en el cálculo, puesto que el logaritmo no está definido para ellos.

Al mismo tiempo, la comprobación respecto a valores muy grandes evita el error de programa que se produciría si el usuario introdujera un número con un valor superior a los que puede almacenar la variable "x", declarada Real. Es precisamente la estructura If la que permite realizar estas verificaciones, que intentan disminuir las posibilidades de que el programa dé lugar a un mensaje no controlado por el programador.

Por desgracia, las verificaciones no contemplan todos los casos posibles. Teclear el programa y comprarlo. Si al escribir el número se produce alguna equivocación y se desliza una letra entre los dígitos, se recibirá un mensaje de error.

Existe otro detalle del programa "Neperiano" que no carece de importancia. Se trata de la colocación en horizontal de las instrucciones. Se sangran (separan del margen izquierdo) de un modo especial. La colocación está pensada para aumentar la claridad del programa y, por tanto, disminuir las posibilidades de error. Aunque en un principio este detalle pueda parecer irrelevante, hay que pensar en lo que ocurrirá cuando se haya leido, o escrito, un programa que contenga unas miles de líneas.

Tal como está escrito el programa contiene básicamente cuatro instrucciones. Las tres primeras son la escritura y lectura, y la estructura de selección o bifurcación.

Al mismo tiempo, también queda muy claro en esta última estructura cuáles son sus partes, cuál es el grupo de instrucciones que se han de realizar, si es cierta la condición (parte Then) o si no lo es (parte Else). A su vez, las dos instrucciones simples que forman la compuesta de la parte Then se detectan al primer golpe de vista. A todo ello ayuda el sangrado escogido. Hay que dedicarle tiempo a la apariencia del código, ya que éste resulta despreciable comparado con el que habría que perder localizando un error que se podría haber evitado con un sangrado mejor.

Después de explicar las instrucciones compuestas, hay que realizar una nueva aclaración sobre la delimitación de los finales de instrucción con los puntos y comas. En las estructuras complejas, éstos no han de escribirse detrás de instrucciones que el compilador puede dar por acabadas con una palabra reservada. Por ejemplo, la instrucción etiquetada con {6} no acaba con ";" porque después se encuentra la palabra reservada End. A su vez, a este End le debería seguir el ";" si no tuviera la estructura la parte Else, porque en ese caso sería el final de la instrucción If. En el programa "Neperiano" el signo ";" que finaliza la estructura If es el que se encuentra al final de la línea etiquetada con {7}.

ELECCIONES MÚLTIPLES

Las elecciones de camino que se pueden tomar en un programa escrito con el lenguaje Pascal pueden ser múltiples.

Uno de los dos modos que existen en este lenguaje para lograr la bifurcación múltiple es encadenar estructuras If siguiendo la sintaxis:

```
If Valor1
 Then ins1
  Else If Valor2
   Then ins2
    Else If Valor3
     Then .....
```

Con este encadenamiento se consigue realizar "ins1", o "ins2", o..., dependiendo de los valores que tomen "Valor1", "Valor2",... En el programa "Traduce1" (figura 4) se encadenan los If para lograr que la aplicación

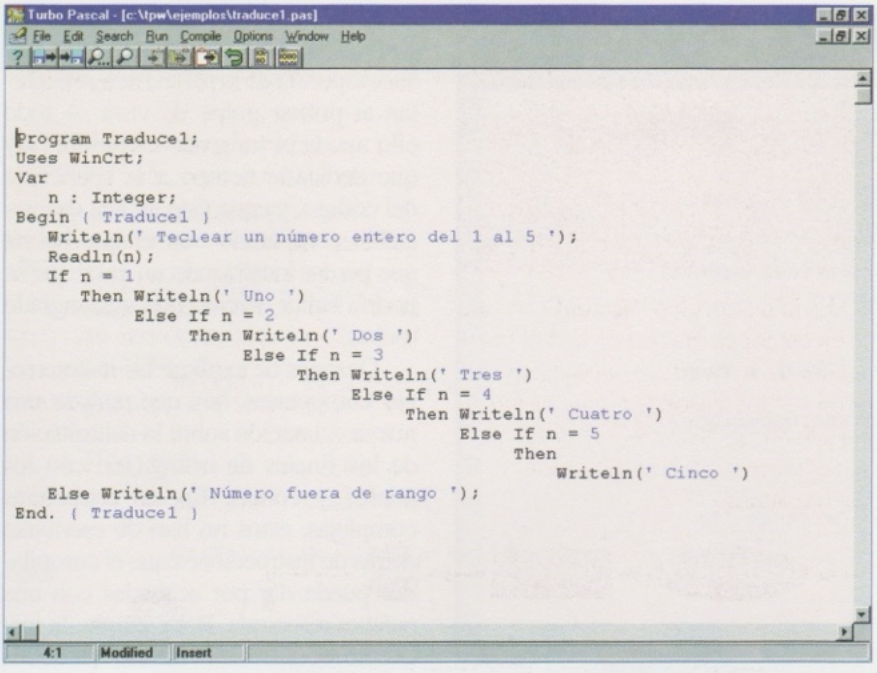


Fig. 4. Selección múltiple en la que se contemplan cinco posibilidades.

traduzca a letras el número que el usuario teclee, siempre que se trate de un número del 1 al 5.

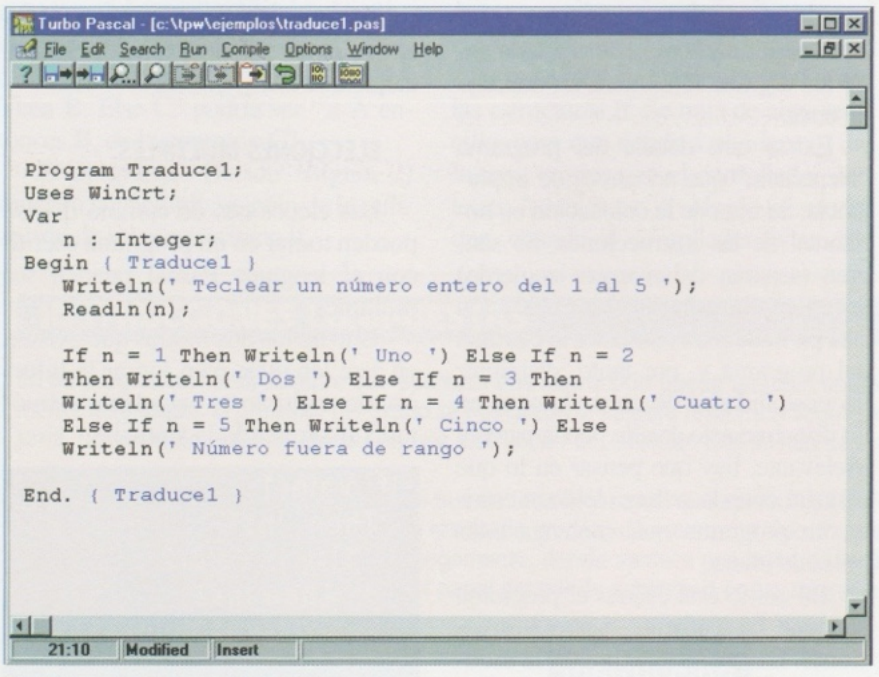


Fig. 5. La selección múltiple es difícil de comprender sin la adecuada ordenación del código.

De entrada, merece la pena señalar que un sangrado razonable ayuda mucho a entender qué está haciendo el programa. En la figura 5 puede ver el mismo programa con una escritura que no tiene en cuenta la claridad. Para entenderlo mejor, comparar las diferencias.

También resulta más fácil entender en el programa de la figura 4 que en el de la 5 el papel relevante que juega la instrucción después del último Else. Es la instrucción que se ejecutará en el caso de que no se cumpla ninguna de las condiciones que se examinan.

Ahora se va a considerar otra posibilidad en la que seguramente el lector ya habrá pensado para modificar el programa "Traduce1" y cuyo resultado será el programa "Traduce2" (figura 6).

En esta nueva versión los If no se encadenan, cada uno es una instrucción independiente, no existen partes Else, y el resultado es el mismo que el obtenido con "Traduce1".

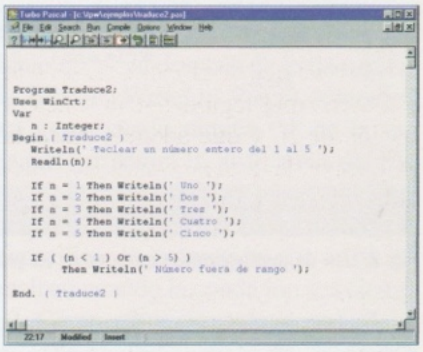


Fig. 6. En este programa no se realiza directamente la selección múltiple.

Hay que comprobar la diferencia existente entre una secuencia de If y un grupo de If encadenados por la parte Else.

Cuando se ejecuta el programa "Traduce2", se comprueba la igualdad del valor almacenado en "n" con los cinco posibles casos, independientemente de que se haya acertado con el valor 1.

Sin embargo, en el programa "Traduce1", al estar encadenados los If por la parte Else, se consigue que dejen de realizarse las comprobaciones en cuanto se haya verificado una de las condiciones. De hecho, si el valor almacenado en "n" resulta ser 1, sólo se realiza la primera comprobación.

No es difícil apreciar que el programa "Traduce1" es más eficiente que el "Traduce2". En el segundo siempre se realizan siete comprobaciones del valor de "n", mientras que el número de las que se realizan en el primero depende del valor que suministra el usuario (en promedio serán tres comproba-

ciones). Pero, aparte de estas consideraciones de eficiencia, hay que resaltar que la diferencia más importante entre ambos está en su legibilidad. En "Traduce1" se utiliza una estructura de selección específica para opciones excluyentes, mientras que en "Traduce2" el que sean exclusivas las opciones queda relegado a las comprobaciones.

En el caso particular de que la acción que haya de realizarse dependa del valor que tome un dato ordinal (datos tipo Char o cualquiera de las variantes del Integer), existe una estructura específica para realizar la selección múltiple.

Se trata de la estructura Case, que es mucho más legible que la concatenación de If, y que además presenta la ventaja de que el compilador puede producir a partir de ella un código máquina más eficiente.

La sintaxis que se ha de seguir en esta nueva estructura es:

```
Case Iden Of
    caso1 : accion1
    caso2 : accion2
      . . .
End ;
```

donde "Iden" es una variable que almacena un ordinal (Char, Integer,...), "accion1", "accion2",... son instrucciones sencillas o compuestas, y "caso1", "caso2",... son la especificación de los valores que ha de tomar en cada caso la variable "Iden". Con esta estructura se consigue realizar la instrucción "accion1" si "Iden" toma el valor "caso1", "accion2" si toma el valor "caso2",... El End se utiliza para indicar el final de la relación de posibles valores.

En el caso de que el valor que tome "Iden" no sea ninguno de los contemplados, pasará a ejecutarse la instrucción siguiente al End. Pero ésta se llevará a cabo también aunque el valor coincida con alguno de ellos. Para lograr que se ejecute una instrucción sólo en el supuesto de que no sea ninguno de los casos contemplados, ha de utilizarse la siguiente modificación de la estructura Case básica:

```
Case Iden Of
    caso1 : accion1
    caso2 : accion2
      . . .
Else
    accionD
End;
```

Es la instrucción "accionD" la que se ejecutará en el caso de que el valor de "Iden" no esté contemplado en ninguno de los casos. La traducción al castellano de "Case A Of B Else C" sería "en el caso de que A entonces B, de lo contrario C".

El programa "Traduce3" (figura 8) produce el mismo resultado que "Traduce1" y "Traduce2", pero es mucho más legible y eficiente.

Para apreciar todas las posibilidades que ofrece la estructura Case, hay que tener en cuenta que los valores que han de ocurrir en cada caso se pueden especificar de un modo bastante amplio. Por ejemplo, si una de las acciones se ha de emprender cuando la variable toma uno entre varios valores, todos ellos se pueden especificar seguidos de comas. En el caso de que este conjunto de valores sea consecutivo, se puede especificar el rango

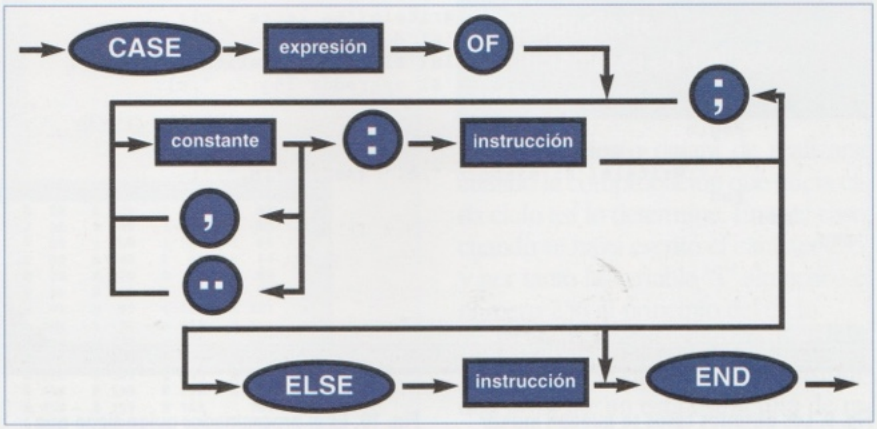


Fig. 7. Gráfico sintáctico de la instrucción Case.

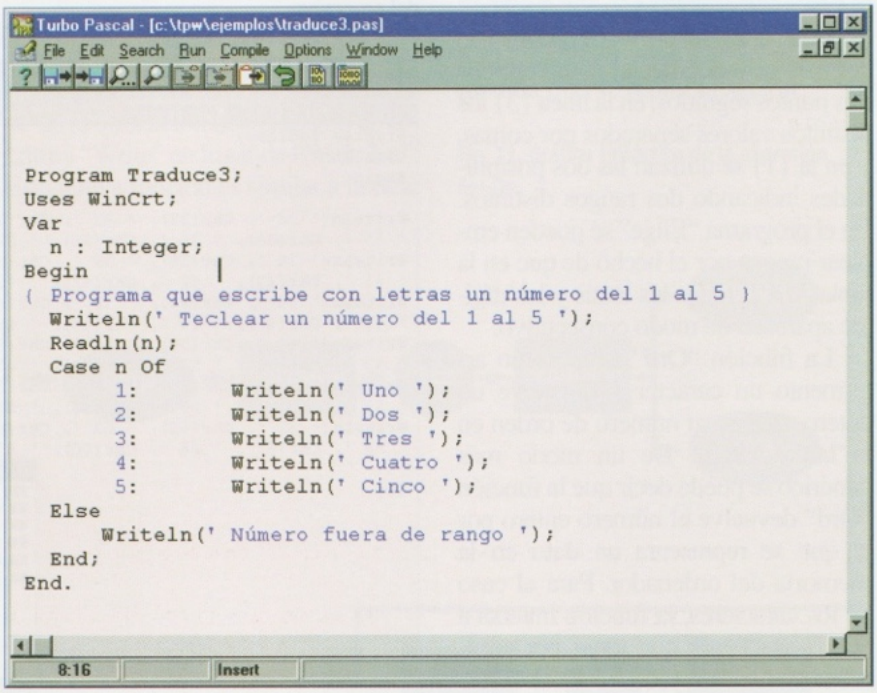


Fig. 8. La estructura Case permite escribir las selecciones múltiples de un modo mucho más claro.

con el primer y último valor unidos por dos puntos seguidos.

El uso más completo de la estructura Case se ilustra en el programa "Elige" (figura 9).

Como ejercicio práctico se sugiere escribir el programa "Elige", modificándolo para así identificar con palabras tanto los paréntesis como los signos de puntuación.

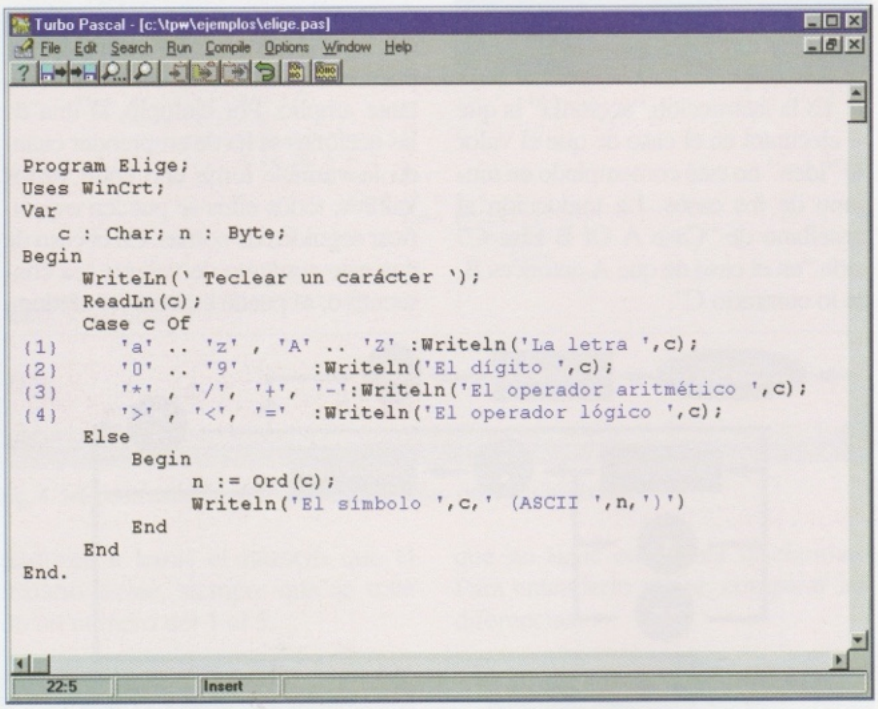


Fig. 9. Los distintos casos se pueden indicar utilizando rangos.

En la línea del programa etiquetada con {2} se especifica el rango con los dos puntos seguidos, en la línea {3} los distintos valores separados por comas, y en la {1} se utilizan las dos posibilidades indicando dos rangos distintos. En el programa "Elige" se pueden emplear rangos por el hecho de que en la tabla ASCII todas las letras y los dígitos aparecen de modo consecutivo.

La función "Ord" tiene como argumento un carácter y devuelve un entero, que es su número de orden en la tabla ASCII. De un modo más genérico se puede decir que la función "Ord" devuelve el número entero por el que se representa un dato en la memoria del ordenador. Para el caso de los caracteres, la función inversa a "Ord" es "Chr", que toma por argumento un entero y devuelve el carácter ASCII que se representa con ese ordinal.

INSTRUCCIONES DE REPETICIÓN

La programación es una tarea que nunca debe ser repetitiva, puesto que el buen programador es siempre capaz de hacer que sea el ordenador quien se encargue de todas las repeticiones. Por ejemplo, supongamos que se desea construir un programa por el que se visualice en pantalla la equivalencia para los caracteres más usados de la tabla ASCII, evitando así el cansancio que supone ir a buscar continuamente el número de orden de cada carácter. Se trata de los comprendidos entre el 32 y el 255.

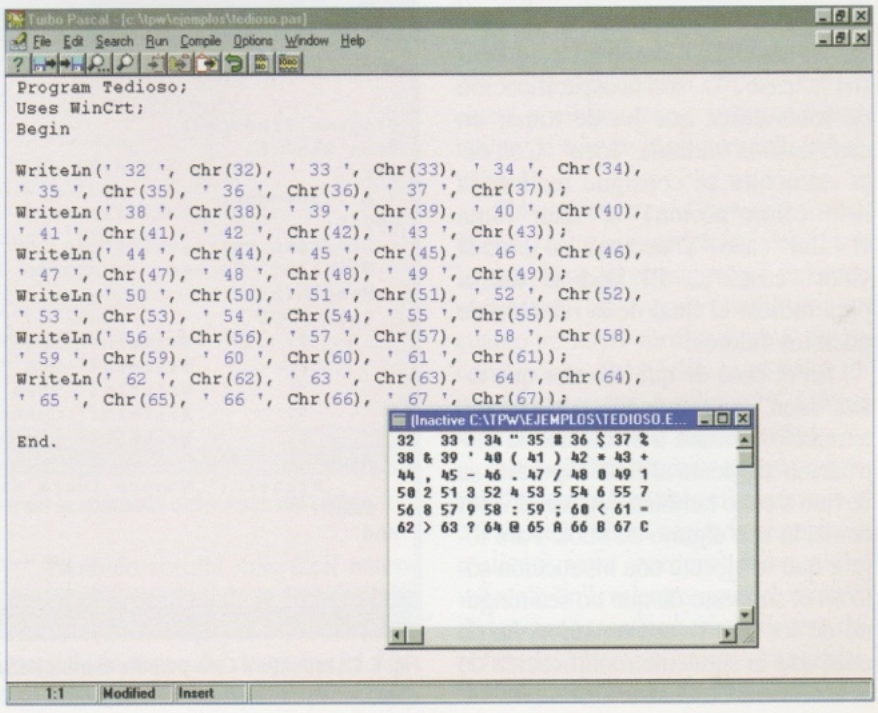
Para ello, se puede utilizar la función "Chr" descrita anteriormente. Así,

```
WriteLn(' 65 ', Chr(65));
```

escribe al lado del número 65 la letra A, que es el carácter con ese número de orden en la tabla ASCII. Como ejercicio práctico, y con mucha paciencia, se puede escribir un programa como el que se muestra, en parte, en la figura 10.

En la esquina inferior derecha de la pantalla también se puede ver la ventana con la salida que produce el programa.

Fig. 10. Es el programador quien tiene que repetir el mismo elemento en muchas partes del programa.



REPETICIÓN FOR-DO

Aparte de la paciencia necesaria para escribir un programa así, las posibilidades de cometer un error son muy grandes. Pues bien, el buen programador hará que sea el ordenador el que realice repetidamente la misma tarea.

El programa "Lista" que aparece en la figura 11 es un buen ejemplo de cómo reducir el trabajo del programador.

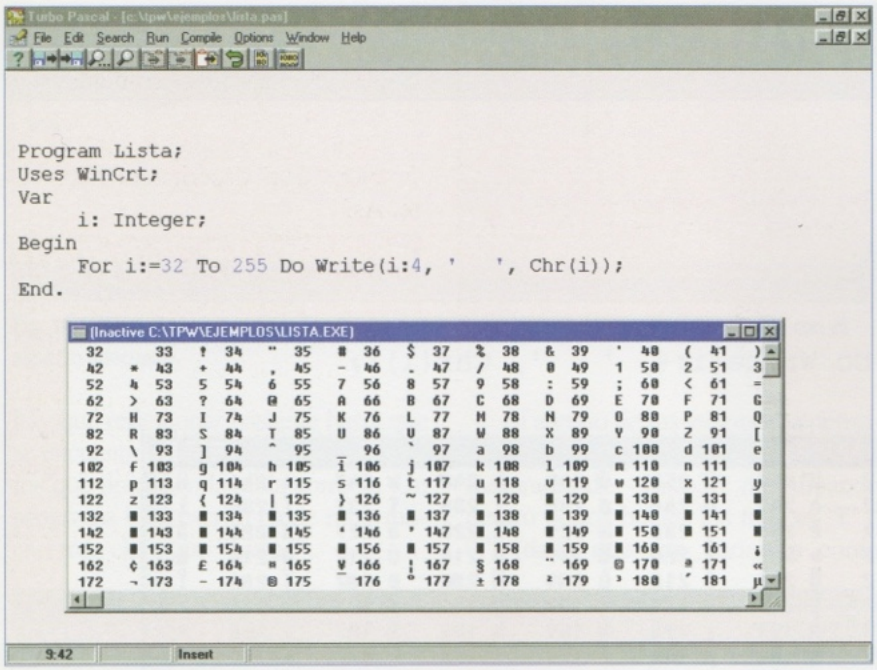


Fig. 11. Una sola instrucción es suficiente para escribir toda la lista.

En una sola instrucción se ordena escribir la lista de los caracteres. En la parte inferior de la pantalla puede verse la salida del programa. En el programa "Lista" se emplea la estructura de repetición For-Do, que ha de utilizarse siguiendo la siguiente sintaxis:

```
For iden := principio To
final Do accion
```

donde "iden" es una variable del tipo Char, Integer, o cualquiera de sus variantes; "principio", el valor que se le asigna a esa variable antes de empezar las repeticiones; "final", el valor máximo que puede alcanzar la variable "iden" antes de acabar las repeticiones; y "accion", la instrucción, simple o compuesta, que se repetirá. Cada vez que se ejecute "accion" se incrementará en uno el valor de "iden". Si se trata de una sentencia compuesta tendrá que utilizarse el par Begin-End para dejar claro el principio y el fin de la instrucción.

Una traducción al castellano de "For A to B Do..." puede ser "desde A hasta B haz...". El gráfico sintáctico de la estructura For-Do se muestra en la figura 12.

Una novedad en el programa "Lista" es la utilización de la función de escritura "Write" en lugar de "Write Ln". Esta nueva función es similar a la otra, pero no incluye el salto de línea cada vez que acaba de escribir los argumentos.

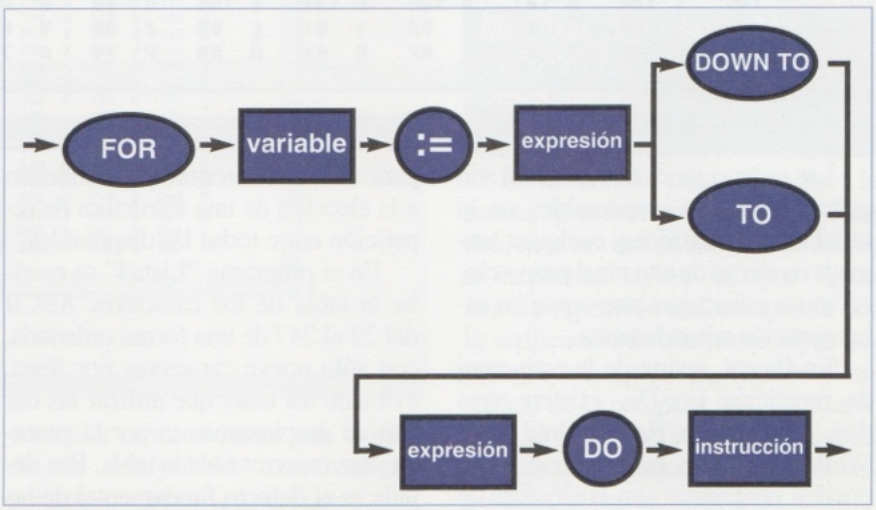
Ahora se van a analizar todas las acciones que implica la ejecución de la estructura For-Do en un caso concreto, el del programa "Lista". Antes de iniciarse las repeticiones, se asigna a la variable "i" el valor 32. Después se ejecutan repetidamente las siguientes operaciones:

- se comprueba que "i" no almacena un valor superior a 255.
- la función "Write" se encarga de escribir el valor almacenado en "i" y el carácter ASCI correspondiente.
- se incrementa en 1 el valor almacenado en "i".

La repetición dejará de realizarse cuando la comprobación que inicia cada ciclo así lo determine. En este caso, cuando se haya escrito el carácter 255, y por tanto la variable "i" almacene el número 256 al principio del ciclo.

El gráfico sintáctico de la figura 12 también contempla otra posibilidad presente en esta estructura de repetición. Es la que se utiliza cuando se desea que el contador que lleva la cuenta de las repeticiones vaya disminuyendo en vez de aumentando a medida que se repite la acción.

Fig. 12. Gráfico sintáctico de la expresión For-Do.



En ese caso la sintaxis es:

```
For iden := principio
Down to final Do accion
```

La traducción al castellano de "For A Down to B Do..." puede ser "desde A disminuyendo hasta B haz ..."

En la figura 13 puede verse la modificación del programa "Lista" para que escriba la lista de caracteres ASCII empezando por el 255 y acabando con el 32.

ellas, pero hay ocasiones en las que una resulta mucho más adecuada que las demás. La elección de una instrucción para una parte del programa se ha de hacer teniendo en cuenta la facilidad de escritura y comprensión de aquél. Por ahora, se dejará para los programadores muy expertos la labor de escribir códigos en los que prime optimizar los recursos del ordenador.

El programa "Lista3" que aparece en la figura 14 sirve como pretexto para analizar criterios que conducen a la elección de una estructura de repetición entre todas las disponibles.

En el programa "Lista3" se escribe la tabla de los caracteres ASCII del 32 al 247 de una forma ordenada, con sólo nueve caracteres por línea, evitando así tener que utilizar las barras de desplazamiento por la ventana para recorrer toda la tabla. Ese detalle es el defecto fundamental de las tablas de caracteres ASCII que se han generado hasta ahora. La salida que produce "Lista3" es el que muestra la figura 15.

El único aspecto positivo del programa "Lista3" es que sirve para ver cómo se anidan dos estructuras For-Do, produciendo una repetición gobernada por dos índices.

El índice "j" varía de 1 a 9 cada vez que aumenta el valor del índice "i". Se realiza este doble bucle para gobernar que en cada línea sólo se escriban 9 caracteres.

Conviene realizar el ejercicio de comprender la utilización del contador "i", pues de ese modo podrá comprobarse que, si bien es sencillo, no resulta obvio. Además, a la hora de realizar un programa de estas características

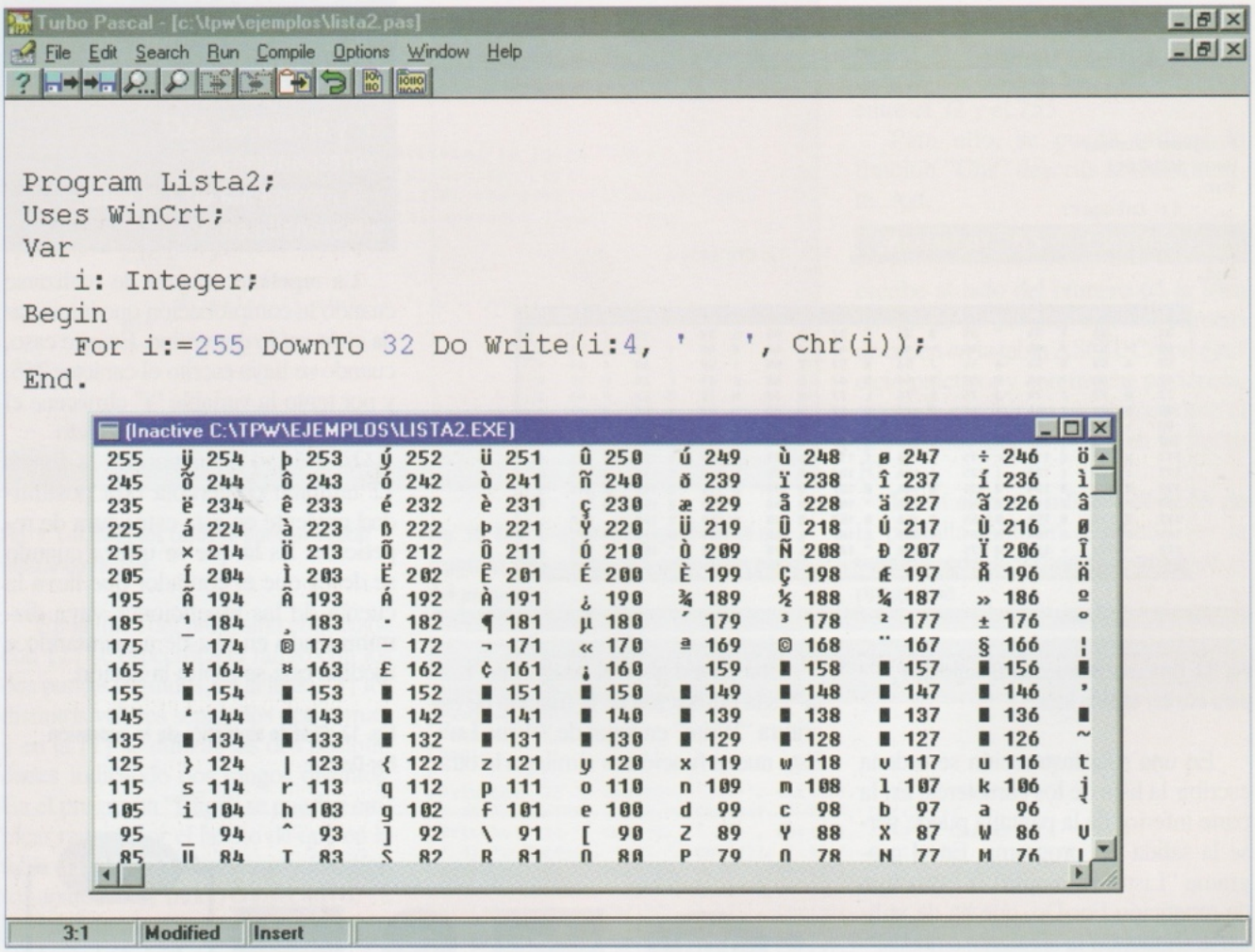


Fig. 13. La tabla de caracteres ASCII escrita de mayor a menor número de orden.

Las instrucciones de repetición son generalmente las responsables de la utilidad del ordenador, y cualquier lenguaje moderno de alto nivel proporciona varias estructuras para controlar esta repetición cómodamente.

En Pascal, aparte de la estructura de repetición For-Do, existen otras dos, que son la Repeat-Until y la While-Do. Todas las repeticiones se pueden programar con cualquiera de

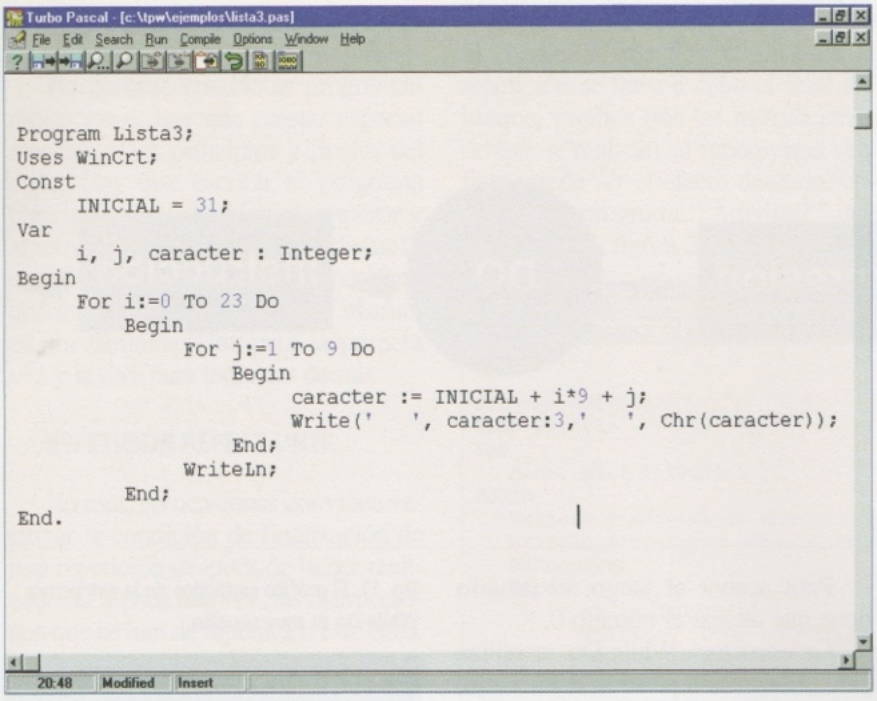


Fig. 14. Utilizando la estructura For-Do no queda muy claro cómo conseguir una tabla con el aspecto adecuado.

hay que reflexionar bastante hasta que se encuentra la combinación de números que producen la tabla correcta. Un programa bien escrito ha de estar mucho más claro que "Lista3".

Si alguno de los lectores ha pensado que podría escribir una versión mejorada de "Lista3" modificando dentro de los bucles los índices "i" o "j" (dado que son variables como

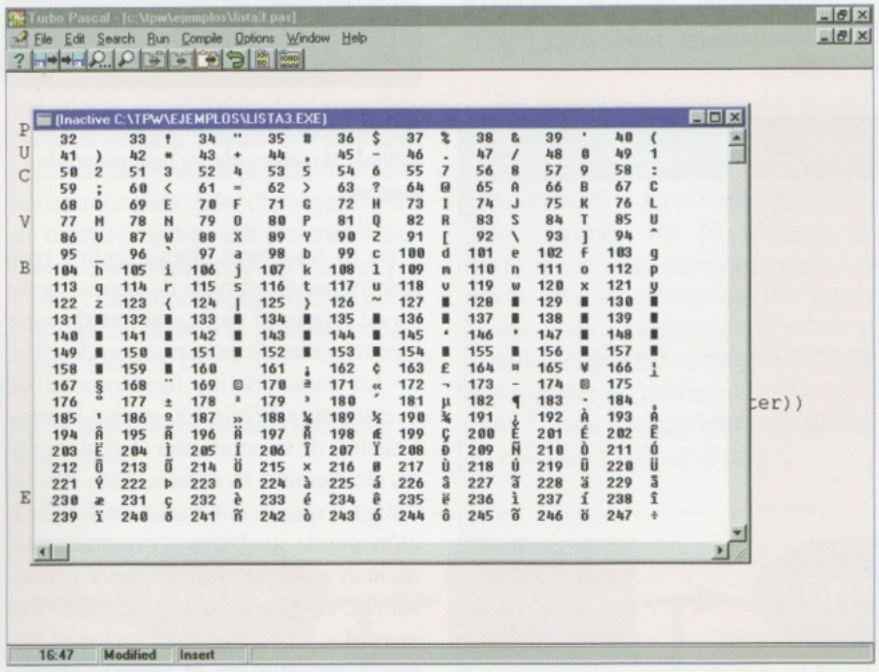


Fig. 15. La tabla que se produce con "Lista3" se lee perfectamente sin desplazamientos por la ventana.

cualquier otra), es mejor que lo olvide. Es una práctica excesivamente peligrosa.

REPETICIÓN WHILE-DO

La verdadera mejora del programa "Lista3" es muy sencilla si se utiliza la estructura de repetición While-Do, tal como se recoge en el programa "Lista4" (figura 16).

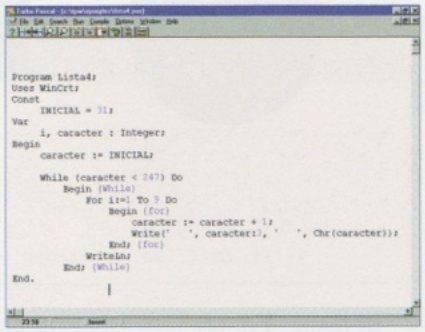


Fig. 16. La estructura While-Do hace más legible el programa que escribe la tabla con el formato adecuado.

La sintaxis de la estructura While-Do es la siguiente:

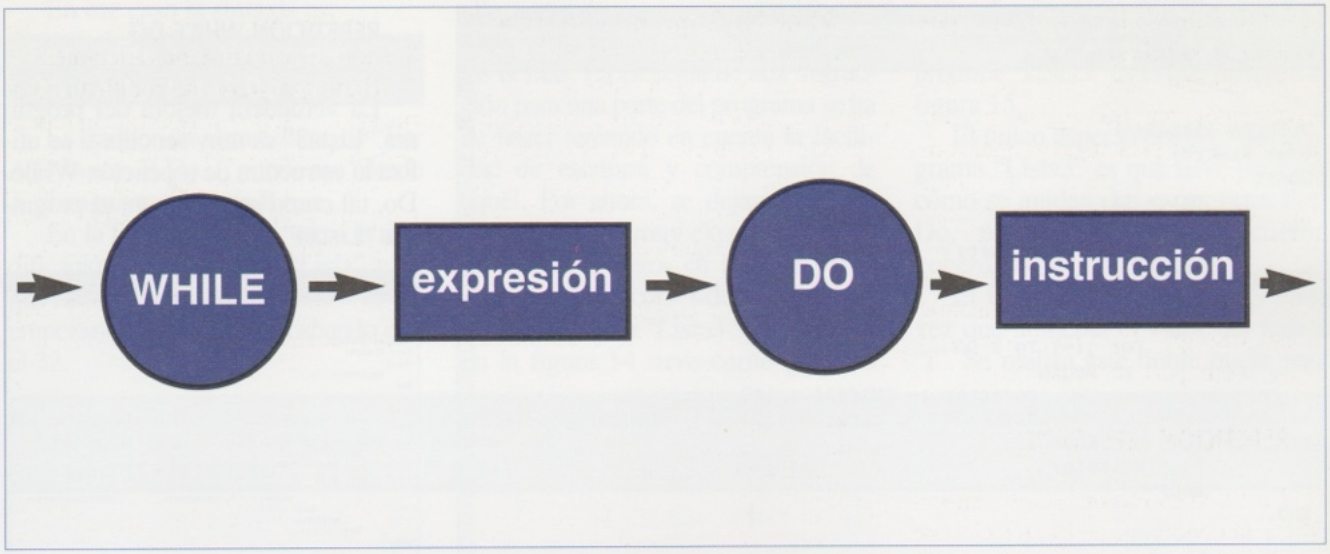
While ValorBoolean Do *accion*

donde "ValorBoolean" es una expresión cuyo resultado es un dato Boolean, y "accion" es una instrucción (sencilla o compuesta) que se repetirá mientras "ValorBoolean" tome el valor True. La traducción al castellano de "While A Do B" es "mientras A, haz B". En la figura 17 se muestra el gráfico sintáctico de esta instrucción.

Utilizando esta nueva estructura se ha escrito el programa "Lista4", que sigue el siguiente pseudocódigo:

Mientras "caracter" <247
- escribir los nueve siguientes caracteres
- saltar a la línea siguiente

En la variable "caracter" se lleva la cuenta de cuántos se han escrito. Para ello, cada vez que se escribe uno nuevo se incrementa en uno su valor (figura 16). La repetición está controlada por la verificación de que no se hayan escrito ya 247 caracteres. Cuando el resultado de la comparación "caracter < 247" se evalúe a False, no se seguirá realizando el bloque de instrucciones gobernadas por el While-Do.



El bucle "For i:= 1 To 9 Do" es el que rige la repetición en cada línea.

Instrucciones de incremento como "caracter := caracter + 1" se utilizan frecuentemente para aumentar el valor de contadores en estructuras While-Do.

Para explicar mejor el uso de esta nueva estructura se ha escrito un programa distinto, el "Adivina1", que aparece en la figura 18.

Se trata de un juego en el que el usuario trata de adivinar el número (entre el 1 y el 5) que ha "pensado" la máquina.

Para acabar el juego, el usuario tiene que teclear el número 0.

La estructura While-Do se utiliza para ordenar la repetición del acertijo, "mientras" el usuario no se aburra y teclee el 0. El ordenador "piensa" el número utilizando la función "Random", que genera secuencias de números denominadas pseudo-aleatorias. La función "Random(n)" devuelve un número entero mayor o igual que cero y estrictamente menor que "n".

El programa "Adivina1" se puede resumir en pseudocódigo del siguiente modo:

Fig. 17. El gráfico sintáctico de la estructura While-Do es muy sencillo.

- escribir cabecera
- inicializar el generador de números aleatorios
- leer número inicial
- mientras no sea 0 el número leído, repetir:
- generar un número aleatorio
- comparar con el leído
- informar de la comparación
- leer un número

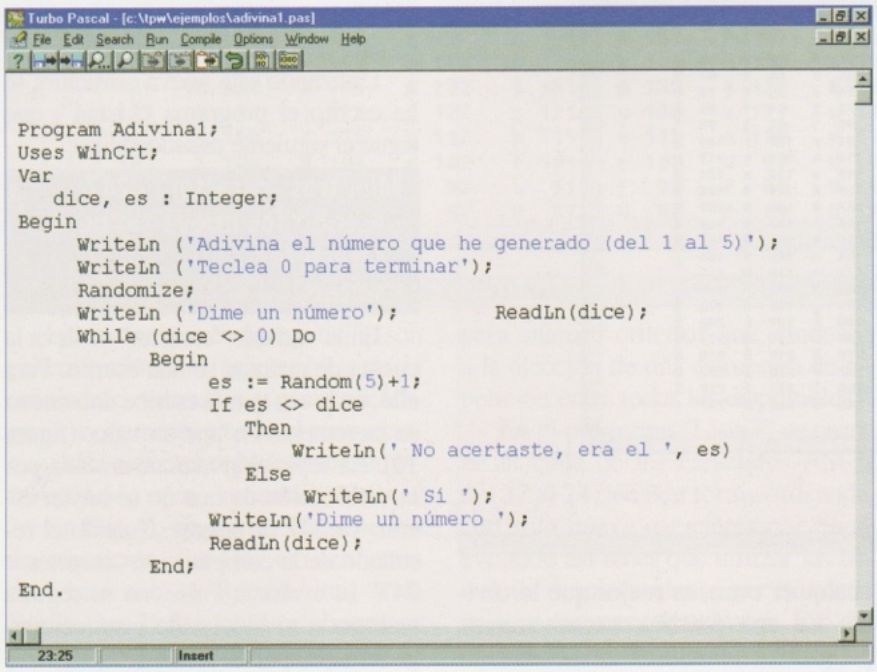


Fig. 18. Juego en el que el usuario trata de adivinar el número que ha determinado la máquina.

y no es necesario ningún índice para determinar el número de veces que se ha de repetir el ciclo, que puede estar ejecutándose un gran número de ellas.

La característica fundamental de la estructura While-Do es que la verificación de la condición se realiza antes de ejecutar la acción. Este detalle obliga a que la condición de finalización pueda evaluarse correctamente, incluso antes de realizarse por primera vez el bloque que se repite. Por ejemplo, en "Adivina1" se lee el número que sugiere el usuario dentro del bloque de instrucciones que se repiten, pero también es necesario leerlo antes de entrar en el bucle para que

la verificación de la condición se pueda realizar la primera vez.

En general, cuando se programan repeticiones, hay que prestar especial atención a los principios y finales del ciclo. Hay que escribir el programa para que todo vaya bien al empezar y al terminar. En el caso de "Adivina1" esto lleva a la necesidad de especificar la lectura del número en dos instrucciones distintas. Una para la primera vez y la otra para todas las demás.

REPETICIÓN REPEAT-UNTIL

En muchas ocasiones conviene verificar la condición de finalización de una repetición después de haber realizado, al menos una vez, las instrucciones que se han de repetir. En este caso, lo apropiado es utilizar la estructura Repeat-Until.

Se emplea con la sintaxis:

```
Repeat accion Until
ValorBoolean
```

y con ella se programa que se repita la instrucción "accion" mientras que no sea True el valor de "ValorBoolean". La traducción al castellano de "Repeat A Until B" es, "repite A hasta que B". En el caso de desear que "accion" sea una instrucción compuesta, no hay que incluir el par Begin-End para delimitar el bloque, puesto que el par de palabras reservadas Repeat-Until son suficientes para dejar claro que se han de repetir cuantas instrucciones sencillas se encuentren entre el Repeat y el Until. Cualquier ambigüedad se podrá resolver acudiendo al gráfico sintáctico de la figura 19.

Fig. 19. Gráfico sintáctico de la instrucción Repeat-Until.

El hecho de que la verificación de la condición que rige el final de la repetición se lleve a cabo al final del bloque, implica que las instrucciones de éste se realizan, al menos, una vez. Esto puede ser el efecto deseado, como en el programa "Adivina2" que aparece en la figura 20.

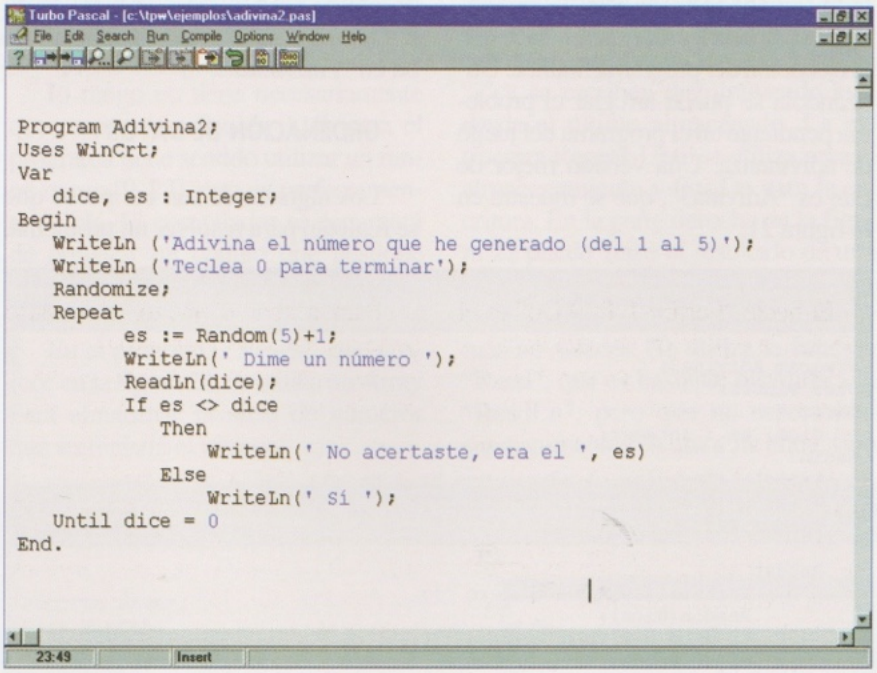


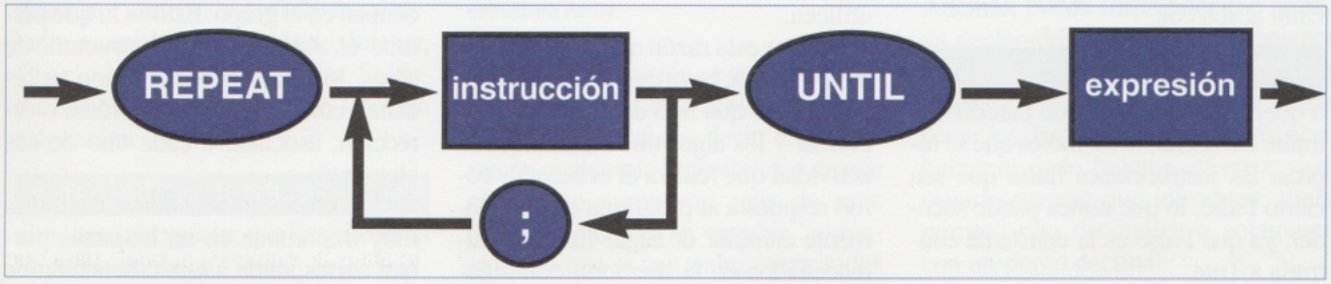
Fig. 20. El programa del juego de adivinanza utilizando la estructura Repeat-Until.

Es una nueva versión del programa del juego de adivinanza, y en ella no es necesario escribir una instrucción de lectura del número específica para la primera vez. No obstante, todavía no resulta completamente correcta esta programación del juego. Como habrá podido comprobar el usuario al ejecutar el programa, tiene que soportar la frase irónica "No acertaste...", aún después de haber tecleado el 0 para finalizar el juego.

No hay que preocuparse, esta situación se corregirá más adelante.

En un programa se codifican algoritmos y, cuando éstos implican repeticiones, siempre hay alguna de las estructuras de Pascal que resulta idónea para programarlo. Hay que elegir siempre aquélla en la que el algoritmo se pueda expresar de un modo natural. Si se escribe primero el algoritmo en pseudo-código, será muy fácil identificarla.

Si el usuario ha visto alguna vez un programa escrito en el lenguaje BASIC, puede estar preguntándose si en Pascal no existe una sentencia análoga al Go To ("Ir a"), que suele inundar aquellos programas. Pues bien, en Pascal sí existe, pero da lugar a un modo de programación contrario a la estructuración que permiten los lenguajes modernos. El mencionado Go To origina bifurcaciones del flujo del pro-



grama difíciles de interpretar y que son la fuente de numerosos errores.

Una ruptura del flujo del programa, en cierto modo similar a la que produce el Go To, es la que provoca la instrucción que se verá a continuación. Se trata de la función "Exit" de Pascal. Se utiliza para modificar, expeditivamente, el flujo de instrucciones del programa: la llamada a esta función hace que la ejecución del programa finalice. Utilizándola se puede arreglar el problema pendiente en el programa del juego de adivinanza. Una versión mejor de éste es "Adivina3", que se muestra en la figura 21.

Se trata de un modo curioso de expresar que el bloque de instrucciones se repita sin parar. Pero esto no quiere decir que para acabar el juego haya que apagar el ordenador.

El juego se acaba cuando el usuario teclea 0 y en el programa se llama a la función "Exit". No sólo acaba el programa sino que acaba bien: se ha evitado el problema que se mencionaba en "Adivina2".

ORDENACIÓN DE DATOS

Los algoritmos son los pasos que se realizan para resolver un problema.

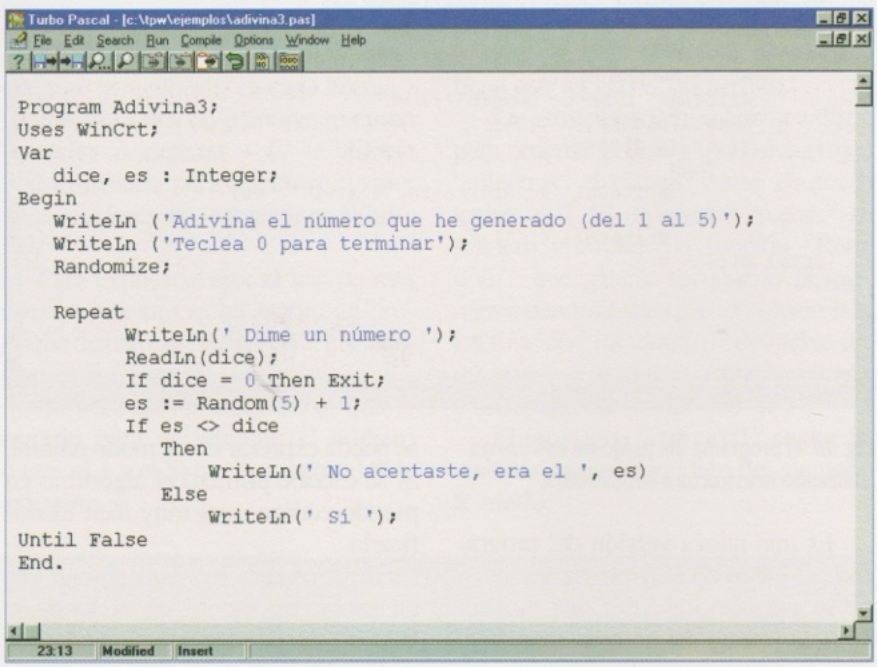


Fig. 21. El programa Adivina3 acaba en el momento adecuado gracias a la llamada a la función "Exit".

Las modificaciones que se han introducido en esta nueva versión parten de "Adivina2" y están pensadas para modificar la condición de finalización de la repetición. La estructura Repeat-Until acaba con

```
Until False;
```

o que, en principio, puede parecer extraño. De hecho, lo es: indica que se repitan las instrucciones hasta que sea cierto False, lo que nunca puede suceder, ya que False es la constante contraria a True.

Y los programas son la traducción a un lenguaje determinado de estos algoritmos. Pero cómo se concreta el algoritmo en cada caso depende enormemente de los tipos de datos que se utilicen.

Es por esta razón que, para diseñar un programa, haya que decidir al mismo tiempo qué tipo de datos se van a utilizar y los algoritmos. Dado que la actividad que realiza el ordenador como respuesta al programa es sencillamente cambiar de lugar los datos almacenados en la memoria con algunas operaciones intermedias, adquiere una enorme importante la forma de estar almacenados aquéllos. Es decir, cómo se agrupan para referirse a ellos.

Hasta ahora sólo se han estudiado tipos de datos básicos de Pascal que carecen de estructura. En la unidad 18 se llevará a cabo un estudio más detallado de cómo el programador puede definir completamente la estructura que tiene los datos en la memoria del ordenador. Ahora se explicará el más básico de los tipos de datos con estructura interna que se pueden utilizar en el lenguaje Pascal. Se trata de uno en el que se agrupan, ordenadamente, una colección de otros datos simples del mismo tipo.

Son los datos denominados genéricamente Array en inglés. Su traducción al castellano es difícil de concretar en una sola palabra, ya que podría ser ordenación, arreglo o tabla, pero también ninguna de ellas. A partir de este momento, para hacer referencia a estos datos, se utilizará el nombre inglés Array que sirve para su declaración en el Pascal, o el término Tabla, que resulta adecuado si se tiene presente que se trata de una tabla en el sentido genérico (igual puede contener una o muchas columnas).

Este nuevo tipo de dato se utiliza para almacenar otros que disponen de una ordenación natural, como los que normalmente se sitúan en tablas, y son especialmente útiles en tareas repetitivas. Primero se explicarán los Array en que los datos se ordenan en una sola dimensión, es decir, como tablas de una sola fila o una sola columna.

Cuando se tiene que seguir la pista a un grupo de datos relacionados, es conveniente referirse a todos ellos con el mismo nombre, y distinguir entre los elementos mediante el lugar que ocupan en el grupo. Esto es lo que permite el Array. En esta estructura de datos, los elementos del grupo se ordenan con un número de orden, o dirección, asociado a cada uno de sus elementos.

La estructura de datos Array es muy importante en un lenguaje, porque corresponde a una ordenación de

datos similar a la que se realiza en la memoria del ordenador.

Para entender mejor el nuevo tipo de dato se puede pensar en un conjunto de cajas contiguas; en cada una de ellas puede almacenarse un dato simple y, para identificar cada uno de los valores almacenados, se utiliza el nombre del conjunto y la posición que ocupa la caja dentro de la fila.

Para definir el tipo de dato Array se emplea la siguiente sintaxis:

```
Array [ rango ] Of tipo
```

donde "tipo" es el identificador del tipo de dato que se almacena en cada una de las posiciones de la tabla, y "rango" la declaración del rango de variación del índice que identifica las posiciones de los elementos de la tabla. El rango se concreta con el primer y último índice separados por dos puntos, uno al lado del otro. Sólo se permite un rango consecutivo. Por ejemplo, en:

```
Var
linea : Array [0..79] Of Char;
```

se declara que la variable "linea" es una tabla en la que se almacenarán datos del tipo Char. En total se pueden almacenar 80 caracteres que se ordenan con un índice que varía entre 0 y 79.

La operación básica que se realiza con un Array es guardar información en una de sus posiciones o recuperarla. Para referirse a un elemento de la ordenación hay que utilizar su nombre y, entre corchetes, el índice que lo identifica. Por ejemplo, "linea[0]" es el primer carácter que se almacena en la fila "linea", "linea[2]" es el tercer elemento, y "linea[79]" el último.

El índice que se sitúa entre los corchetes ha de ser, lógicamente, un ordinal, por tanto se tratará de una constante o variable del tipo Byte, Integer, Char, o cualquier otra similar. Si "letra" es una variable del tipo Char, la asignación

```
linea[10] := letra;
```

almacena el carácter que se guarda en el grupo de datos "linea" en la posición con índice 10 (de hecho, la 11 desde el principio del Array "linea"). La operación inversa para recuperar la información guardada es:

```
letra := linea[10] ;
```

A la hora de decidir el rango de índices que hay que utilizar para una variable del tipo Array tiene que primar la lógica impuesta por el algoritmo que se esté programando.

El rango no tiene necesariamente que empezar en 0 o en 1. Si para el programa tiene sentido utilizar un rango como [9..23], éste es perfectamente válido. El compilador se encargará de preparar un código que gestione adecuadamente la memoria del ordenador.

En el programa "Reves", que aparece en la figura 22, se utiliza un Array para almacenar la serie de números que suministra el usuario.

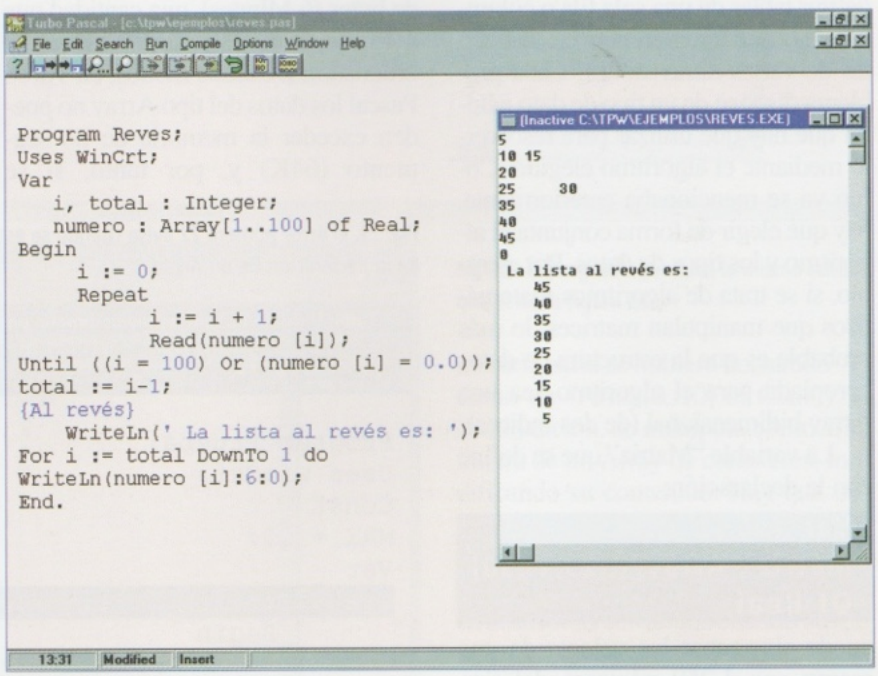


Fig. 22. Ejemplo del uso del tipo de dato con estructura Array.

En la declaración se especifica que se pueden almacenar hasta 100 números del tipo Real. Es decir, el programa no contempla la posibilidad de que el usuario suministre más de esos números, pues en la variable "numero" en la que se van a almacenar todos no caben más de 100. El programa "Reves" lee los números que le suministra el usuario hasta que teclee un cero. Una vez que esto haya sucedido, los escribirá todos en orden inverso al utilizado para suministrárselos. Para retenerlos en la memoria se utiliza el Array "numero" y se pueden obtener ordenadamente empleando el índice que identifique su posición. Se almacenan en orden creciente del índice "i" y se escriben disminuyendo éste desde el último almacenado. La estructura Repeat-Until se utiliza para el almacenamiento y For-Do para la escritura. En la parte derecha de la figura 22 puede verse el resultado de una sesión de uso del programa.

Pero en el programa "Reves" hay más novedades. Se utiliza la función "Read", que es bastante parecida a la "ReadLn", pero que no espera leer sus argumentos en una sola línea, sino que intenta hacerlo en una o varias. Además, puede sorprender que se haya declarado un Array para contener datos Real. Pues bien, quizás podrían haber servido los Integer para almacenar los datos en muchos casos, pero hubieran dado lugar a un error si el usuario tuviese que introducir algún número con decimales o solamente con un punto decimal.

Cuando se declara una variable del tipo Array, aparte de informar al compilador de su significado, hay que hacerlo de la cantidad de memoria que debe reservar para ella. Ésta viene determinada por el rango de índices. Por tanto, cuando el programador escribe el programa "Reves" tiene que decidir cuál será el número máximo de datos que va a almacenar en la variable y reservar memoria para el peor de los casos. La pregunta que surge es si no existe la posibilidad de postergar la decisión para el momento de la ejecución, cuando el usuario decida el uso que va a hacer del programa. La respuesta es no, si se utilizan Array. Para que sea posible conseguir este efecto hay que proceder a programar lo que se denomina asignación dinámica de memoria, una parte del lenguaje Pascal que todavía no se ha explicado.

En Pascal no solamente es posible definir tablas de una sola fila o columna, sino que también permite definirlas de varias dimensiones. Cada problema dispone de un tipo de dato natural que hay que utilizar para resolverlo mediante el algoritmo elegido. Como ya se mencionaba anteriormente, hay que elegir de forma conjunta el algoritmo y los tipos de datos. Por ejemplo, si se trata de algoritmos matemáticos que manipulan matrices, lo más probable es que la estructura de datos apropiada para el algoritmo sea una Array bidimensional (de dos índices).

La variable "Matriz" que se define con la declaración:

```
Var
Matriz : Array[1..25,1..50]
Of Real;
```

puede almacenar los valores de una matriz con 1.250 números del tipo Real (ocupando 7.500 bytes en la memoria del ordenador). A todos estos números se accede con dos índices. Por ejemplo, con

```
matriz[3,7] := 100.0;
```

se guarda el valor 100.0 en la posición [3,7] de la tabla bidimensional "matriz". Pero hay un detalle relacionado con la memoria del ordenador que nunca se debe olvidar al utilizar tablas con varias dimensiones: la cantidad de elementos que caben en una tabla multidimensional crece rápidamente con el rango máximo de los índices. Si se utiliza un tipo de variable como la siguiente:

```
Var
grande : Array[1..MAX,
1..MAX,1..MAX] of Real;
```

hay que tener en cuenta que el número de elementos que puede almacenar es MAX*MAX*MAX, y la cantidad de memoria, en bytes, que necesita el ordenador para ello la obtiene multiplicando ese número por el de bytes necesarios para guardar cada elemento. En el caso del ejemplo, son 6 bytes para cada elemento. Así que, si a la constante MAX se le da el valor 100, la memoria que se reservaría para la variable "grande" sería de 6 millones de bytes (6 Mbytes), una cantidad que a lo mejor no está disponible en el ordenador. De todos modos, en Turbo Pascal los datos del tipo Array no pueden exceder la memoria de un segmento (64K) y, por tanto, si se declarara "grande" con un valor de MAX superior a 22, el compilador avisaría del error con el mensaje "Structure too large" (estructura demasiado grande), indicando que se está excediendo la capacidad máxima.

Por casualidad, este error tiene el número de orden 22.

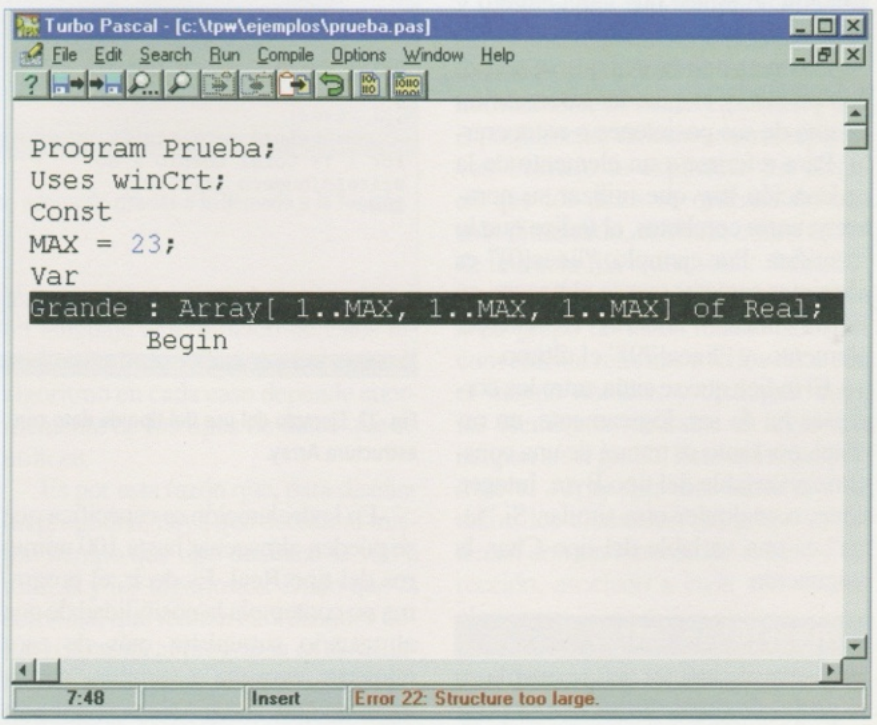
Por supuesto, con Turbo Pascal se pueden declarar estructuras para almacenar datos cuya única limitación sea la memoria disponible en el ordenador, pero éstas no son las Array.

DATOS STRING: SU ESTRUCTURA INTERNA

Antes de explicar las estructuras del tipo Array ya se estaban utilizando, aunque sin saberlo, variables con estructura interna. Se trata de las String que almacenan cadenas de caracteres.

En efecto, los datos del tipo String son un caso particular de los Array: una ordenación de caracteres como la de Array, pero en la que la primera posición se reserva para almacenar un valor que corresponde a la longitud de caracteres almacenados.

Fig. 23. El error número 22 avisa cuando se está excediendo la capacidad de almacenamiento en la declaración de un Array.



Si se parte de la declaración:

```
Var
s : String;
```

se hace referencia a los caracteres almacenados en "s", uno a uno, como si fuera un Array que almacenase hasta 255 valores del tipo Char en las posiciones que van desde "s[1]" hasta "s[255]"; el ordinal del carácter que se almacena en "s[0]" indicaría el número de caracteres almacenados en cada momento.

Por ejemplo, en el programa "Cara" (figura 24) se realizan este tipo de operaciones, al escribir uno a uno los caracteres almacenados en la variable del tipo String. La función "Ord()" devuelve el entero correspondiente al ordinal del carácter que recibe en su argumento. De hecho, lo que hace es traducir a un número decimal la secuencia de ceros y unos que hay almacenadas en la variable que tiene en el argumento. En el caso del programa "Cara" el ordinal almacenado en c será 4, puesto que ése es el número de caracteres almacenados.

La función contraria a "Ord()" es la "Chr()", que devuelve el valor Char con el número de orden correspondiente al entero que recibe como argumento. Por ejemplo, "Chr(65)" devuelve el carácter A (mayúscula). La función "Chr" es la que se utiliza en el programa "Cara2" (figura 25).

En éste se rellena un Array de caracteres, tal como si fuera el String equivalente, pero la diferencia es que, para el dato definido de este modo, no se puede utilizar la asignación global del valor de la cadena.

Han de asignarse uno a uno todos los caracteres.

Es interesante estudiar el programa "Modifica" (figura 26), junto a la salida que produce. En él se declara la variable "s" del tipo String y se rellena utilizando la asignación de toda la cadena que es posible para este tipo de datos (hay que recordar que esto no sería así si se hubiera declarado "s" como un "Array[0..10] of Char"). A continuación, se manipula como si se tratara de un Array de caracteres, modificando su contenido. Hay que observar que al modificar el contenido de "s[0]" se está cambiando la cantidad que indica cuántos caracteres han de considerarse válidos cuando se trate la estructura como un String.

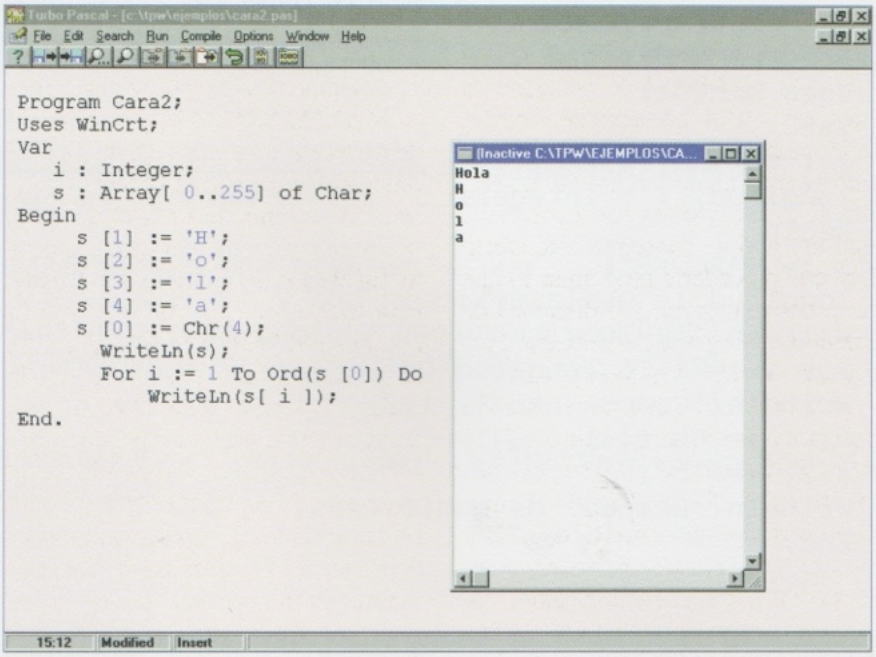


Fig. 25. El programa produce la misma salida que el de la figura 24.

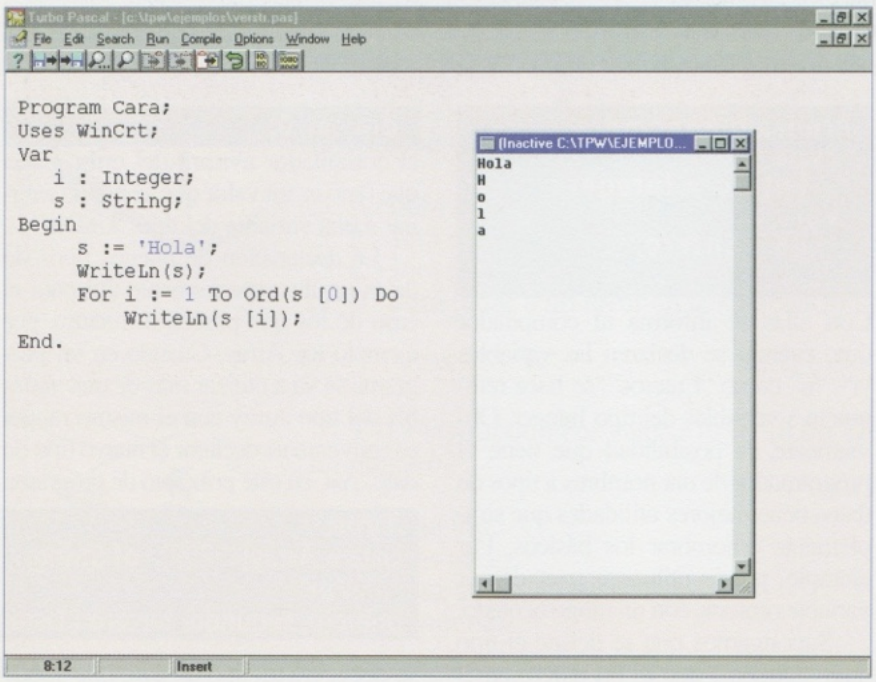


Fig. 24. Se puede acceder individualmente a los caracteres almacenados en un String, tal como si fuera un Array de caracteres.

DAR NOMBRE A LOS TIPOS DE DATOS

Ya que se han empezado a utilizar tipos de variables con estructura, es conveniente saber que Pascal ofrece la posibilidad de nombrar los tipos de datos que define el usuario, lo que mejora la legibilidad de los programas.

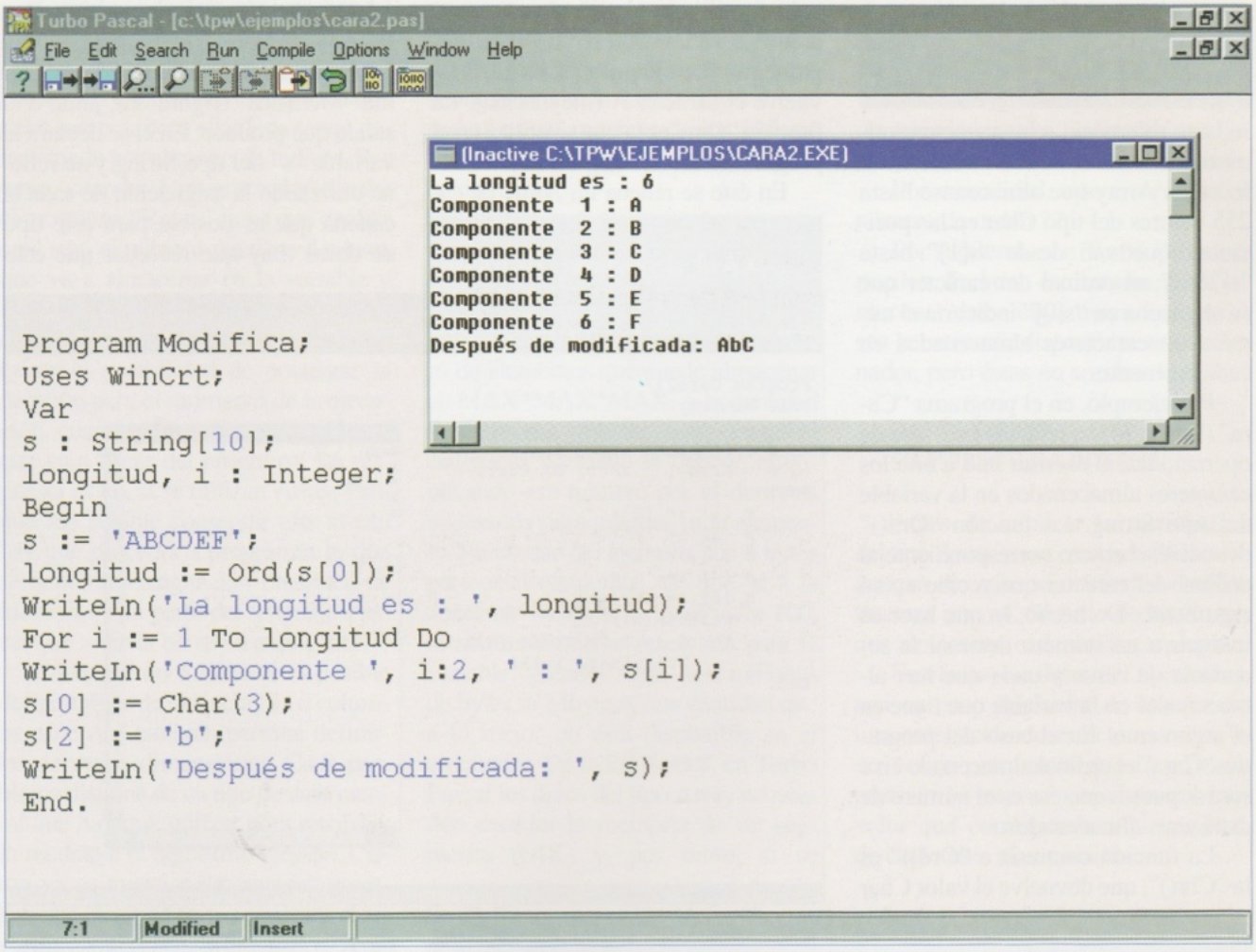


Fig. 26. Al escribir por segunda vez la cadena "s", sólo se escribirán tres letras al ser ese el número de ellas indicado por "s[0]".

Para ello se utiliza en la parte declarativa la construcción Type con la sintaxis:

```
Type
  Iden1 = Untipo1;
  Iden2 = Untipo2;
          . . .
```

donde "Iden1", "Iden2", ..., son los nombres asignados a los nuevos tipos de datos, y "Untipo1", "Untipo2", ..., la especificación de qué tipo de dato se trata.

Por supuesto, la declaración del nombre del tipo de dato tiene que aparecer antes de que se utilice para declarar variables.

El empleo más trivial que se puede pensar para esta posiblidad es el de renombrar los tipos de datos estándar del Pascal. Por ejemplo,

```
Program Renombra;
 Type
   Entero = Integer;
 Var
   i,n : Entero;
   . . . . . .
```

Con ellas se informa al compilador que, cuando se declaren las variables "i", "n" como "Enteros", se hace referencia a variables del tipo Integer. Obviamente, la posibilidad que tiene el programador de dar nombres a tipos de datos tiene mejores utilidades que simplemente renombrar los básicos. Por ejemplo, puede utilizarse para definir variables enteras con un rango acotado.

Supongamos que se define el tipo dato "UnDia", que es un entero en el rango entre 1 y 31. Si, por equivocación, en algún lugar del programa se desliza la asignación

```
dia := 0;
```

el compilador avisará del error, dado que 0 no es un valor que se pueda asignar a una variable del tipo "UnDia".

La declaración de nuevos tipos de datos resulta especialmente útil para el caso de los que poseen estructura, por ejemplo los Array. Cuando en un programa se va a utilizar más de una variable del tipo Array con el mismo rango, es conveniente declarar el nuevo tipo de dato. Así, en este principio de programa

```
Type
Lista = Array[1..200]  of String;
    Var
primero, segundo, tercero : Lista;
```

se declarará un tipo de dato que puede almacenar 200 nombres, cada uno de ellos guardado en un String y se declaran tres variables de este nuevo tipo.

CUESTIONARIO AUTOEVALUATIVO

1 - Después de ejecutarse la instrucción If a = 2 Then b := 3; ¿qué valores se almacenan?

- ❑ A - "a" siempre 2.
- ❑ B - "b" siempre 3.
- ❑ C - "a" dependerá del valor almacenado en "b".
- ❑ D - "b" dependerá del valor almacenado en "a".

2 - Después de ejecutarse la instrucción If (uno And Otro) Then a:= 3; b:= 5; ¿qué valor almacenará la variable "b"?

- ❑ A - 5 independientemente del valor de "uno" y "otro".
- ❑ B - 5 sólo si las variables "uno" y "otro" tienen el valor True.
- ❑ C - 5 si una de las dos variables "uno" y "otro" toma el valor True.
- ❑ D - 5 ó 3, dependiendo de los valores de las variables "uno" y "otro".

3 - En la instrucción If a = 1 Then x:= 10 Else If a = 2 Then x:= 20 Else x := 50; si "a" contiene el valor 10, ¿cuál tomará "x"?

- ❑ A - 1
- ❑ B - 10
- ❑ C - 20
- ❑ D - 50

4 - Si la variable "letra" está declarada del tipo String, ¿qué ocurrirá con la instrucción de bifurcación que empieza Case letra of?

- ❑ A - Dará lugar siempre a un error en la ejecución del programa.
- ❑ B - Será correcta si "letra" almacena una letra.
- ❑ C - Será correcta si "letra" almacena un número.
- ❑ D - Dará lugar a un error en la compilación del programa.

5 - La instrucción For valor:=10 To 10 Do Writeln(valor); ¿cuántas veces escribirá un número?

- ❑ A - Una vez.
- ❑ B - Dos veces.
- ❑ C - Diez veces.
- ❑ D - Ninguna.

6 - La serie de instrucciones a:= 2; Repeat a:= a * a Until (a > 100), ¿qué valor acabarán conteniendo en "a"?

- ❑ A - 16.
- ❑ B - 256.
- ❑ C - 65536.
- ❑ D - 99.

7 - ¿Qué efecto produce la instrucción While False Do WriteLn('False');?

- ❑ A - Es incorrecta sintácticamente.
- ❑ B - Se quedará escribiendo indefinidamente.
- ❑ C - No producirá ninguna salida.
- ❑ D - Escribirá una sóla palabra.

8 - Si la variable "grupo" está declarada como grupo : Array [0..20,1..2] Of Real, ¿cuánto ocupará en memoria?

- ❑ A - 42 bytes.
- ❑ B - 252 bytes.
- ❑ C - 40 bytes.
- ❑ D - 240 bytes.

9 - La variable "linea" declarada como String[10] ¿cuánto ocupará en memoria?

- ❑ A - 0 bytes.
- ❑ B - 11 bytes.
- ❑ C - 9 bytes.
- ❑ D - 20 bytes.

10-Con las declaraciones **Type** serie = Array[1..20] **of Integer; Var** a :serie; ¿cuál de las siguientes asignaciones es correcta?

- ❑ A - a:= 100;
- ❑ B - a[2] := 100;
- ❑ C - a[100]:= 2;
- ❑ D - a:= [2..4];

Respuestas: 1 D - 2 A - 3 D - 4 D - 5 A - 6 B - 7 C - 8 B - 9 B - 10 B